International Journal of Mechanical and Production Engineering Research and Development (IJMPERD)
ISSN (P): 2249-6890; ISSN (E): 2249-8001
Vol. 8, Issue 3, Jun 2018, 327-338

JOB SHOP SCHEDULING USING DIFFERENTIAL EVOLUTION ALGORITHM

P. SHILPA[1] & G. BALARAJU[2]

[1]*Assistant Professor, Department of Mechanical Engineering, ACE Engineering College, Ghatkesar, Hyderabad, Telangana, India*

[2]*Professor, Department of Mechanical Engineering, ACE Engineering College, Ghatkesar Hyderabad, Telangana, India*

ABSTRACT

This paper presents a differential evolution approach for job shop scheduling problems. The job shop scheduling problem, a class of combinatorial optimization problem that involves discrete optimization over discrete variables and the most important objective is makes to pan minimization. The differential evolution approach is a stochastic based effective adaptive scheme for global optimization over continuous space. In order to apply the differential evolution algorithm for the job shop scheduling problem, a suitable encoding mechanism is required to generate an operation schedule. In the present work, the operation schedule is generated using random keys encoding scheme that deals with floating point vectors and a local search heuristic is used to achieve the best optimal solution. The proposed approach is extensively tested on a set of standard job shop scheduling instances and the results are compared with the best- known solutions. It is observed that the proposed approach is performing well on all the test problems.

KEYWORDS: Job Shop Scheduling, Differential Evolution, Random Keys, Makes Pan & Local Search

Received: Mar 20, 2018; **Accepted**: Apr 10, 2018; **Published**: May 03, 2018; **Paper Id.**: IJMPERDJUN201837

Original Article

INTRODUCTION

Increasing changes in the technology and customers demand made an impact on manufacturing strategies and emphasized the need for products in small volume. To stay competitive in this dynamic environment that demands quick delivery of products at low cost, there is a need for effective scheduling of the production activities (Jain, 1999). Hence, most of the research is focused on the job shop scheduling that suits the present day customers demand. Then- job and m- machine job shop scheduling problem (JSSP) is one of the most general production problems. Each job consists, a set of operations and each operation of the job is characterized by a specific machine and its corresponding processing time. The Job characteristics typically impose certain constraints such as precedence relations among the operations of a job, job priorities and job due date's etc., make the problem more complex in nature. The most important objective of scheduling problems is to minimize the total completion time of all the jobs and is referred as makes pan and the other objectives include minimization of mean flow time and mean tardiness (Baker,1974)

Scheduling problems are combinatorial in nature and solving such problems amounts to make a discrete choice of an optimal solution among the finite number of alternatives available (Bagchi, 1999). Decisions involve which job to be scheduled first and when to schedule the next job. There are two types of order relations in the job shop scheduling problem. One is operation sequence on each machine and another is precedence constraints among

the operations of a job. The first one is determined by a solution and the second one is maintained in the schedule.

Most of the scheduling problems are NP- Hard and it is difficult to find an optimal solution without the use of an enumerative algorithm (Cheng et.al, 1996). A heuristic method, that uses a trial and error, at the best, may produce an approximate solution. However, finding near-optimal solutions using approximate algorithms within a reasonable time is acceptable and has become a practice (Carlier and Pinson, 1989). The present work is an attempt to address the job shop scheduling problem using the differential evolution algorithm for make span minimization objective.

LITERATURE SURVEY

The job shop scheduling problem has received the attention from many researchers of various backgrounds for the last two decades. Carlier and Pinson (1989) proposed an efficient branch and bound method that used improved bounding techniques and a new heuristic rule for branch selection. They were the first to prove the optimality of a solution for the 10 jobs and 10 machine problem proposed by Fisher and Thompson (1963). Because of the difficulty of using a branch and bound technique and developing special-purpose heuristics for the job shop scheduling problems, general heuristic search techniques have been applied to these problems in the recent years. Three of the most promising heuristic methods that widely used are genetic algorithms, tabu search and simulated annealing. Biegel and Davern (1990) applied Genetic Algorithms to the job shop-scheduling problem and discussed an elementary n tasks one-machine problem. They extended the work for n tasks on 2 processors and finally generalized for n tasks and m processors. Later Falkenauer and Bouffouix (1991) addressed job shop with many tasks, and many machines using Genetic Algorithm. Nowcki and Smutnicki (1996) applied Tabu search technique and achieved better solutions for job shop scheduling problems. Binato et.al.(2002) described a greedy randomized adaptive search procedure (GRASP) for solving job shop scheduling problems. Bean (1994) proposed a random keys Genetic algorithm for job shop scheduling which produces a feasible schedule without repair during genetic operations. Goncalves et.al (2002) proposed a hybrid genetic algorithm for job shop scheduling problems using a random keys encoding method.

In recent years, memetic algorithms have been used for optimization of scheduling problems in real-world environments. Some recent works on a memetic algorithm by Krasonogar et.al (2004) have shown their performance in global search. Recently, a differential evolution (DE) has been proposed by Storn and Price (1997) for global optimization over continuous spaces. Since its invention, DE has been applied in many numerical optimization problems and moreover different variants of the basic DE model have been proposed to improve the efficiency. Kaelo and Ali (2006) proposed modifications in mutation and local search in DE resulting in two new versions of the original algorithm. Experiments over multiple benchmark problems showed that these new memetic algorithms are performing well in global optimization.

Due to the continuous nature of DE, its application to combinatorial optimization problems with discrete decision variables is rare and unusual. Recently Andreas C. and L. Omirou (2006), proposed a differential evolution approach for flow shop scheduling problems using a random keys encoding scheme.

In the present work, the authors made an attempt to address the job shop scheduling problems using a hybrid DE algorithm. A suitable random key encoding mechanism is proposed to implement the algorithm effectively, and the solution is decoded using a special conversion function.

JOB SHOP SCHEDULING PROBLEM

The classic n x m Job Shop Scheduling Problem is given by a finite set J of n jobs $\{J_i\}$ $1 \leq i \leq n$ and a set M of m machines $\{M_k\}$ $1 \leq k \leq m$. Each job J_i has to be processed on every machine and consists of a chain of m_i operations $\{O_{ik}\}$ $1 \leq i \leq n$, $1 \leq k \leq m_i$ which has to be scheduled in a strictly sequential way according to the given technological order, also referred to as the precedence constraints (Jain,1999). Thus O_{ik} denotes the operation of job J_i that has to be processed on machine M_k for a certain uninterrupted processing time T_{ik}. The other constraint is each machine can process only one job at a time (capacity constraint). The time span required to complete all operations of all the jobs is known as make span (C_{max}) and is the most important objective of JSSP.

Objective criteria: Make span C_{max} = max of $[C_1, C_2, C_3, \ldots\ldots C_n]$ (1)

Where C_1, C_2, C_3 are completion times of Jobs 1,2 and 3

DIFFERENTIAL EVOLUTION ALGORITHM

Evolutionary algorithms (EA) have received an attention in solving a wide range of optimization problems. Evolutionary algorithms mimic the metaphor of natural biological evolution, which adopt changing environments to find the optimum through evolving a population of candidate solutions (Storn, 1997). DE is an exceptionally simple evolutionary strategy that is significantly fast and robust for numerical optimization and can be classified as a class of floating point encoded evolutionary algorithm. The major differences between DE and Genetic algorithm (GA) are in GA, all off springs are accepted and their parent strings are abandoned at the end of every generation regardless of their fitness values. This gives rise to a risk that a good parent string may be replaced with its deteriorated child string. Thus the improvement of the average performance of the child population over parent population cannot be always guaranteed. This does not occur in DE as a child string has to compete with its parent to get a place to participate in the next generation. Moreover, in GA only good parent strings are given the chance to produce off springs without any consideration of the possibilities of generating better offspring by others. In DE, all solutions get the same chance of being selected, without depending on their fitness values. Unlike other EAs, where perturbation occurs in accordance with a random quantity, DE uses a weighted difference between solution vectors (target vectors) to produce population at each generation. These characteristics make DE perform better than other evolutionary algorithms.

DE Strategies

The most crucial and important factor in any heuristic is its internal manipulation routines. DE is highly effective due to its novel and robust internal mutation schema. Price and Storn (1997) described ten different working strategies of DE, which are usually dependent on the problem to be solved. Each strategy is dependent on three factors; the solution to be perturbed, a number of different solutions considered for perturbation and the type of crossover used. The different strategies are given as:

- DE/best/1/(exp/bin) : $\mathbf{U_i = X_{best} + F.(X_{r1}-X_{r2})}$
- DE/rand/1/(exp/bin) : $\mathbf{U_i = X_{r1} + F.(X_{r2}-X_{r3})}$
- DE/rand-to-best/1/(exp/bin) : $\mathbf{U_i = X_i + F.(X_{best}-X_i) + F.(X_{r1}-X_{r2})}$
- DE/best/2/(exp/bin) : $\mathbf{U_i = X_{best} + F.(X_{r1}-X_{r2} - X_{r3}-X_{r4})}$
- DE/rand/2/(exp/bin) : $\mathbf{U_i = X_{r5} + F.(X_{r1}-X_{r2} - X_{r3}-X_{r4})}$

The convention shown is on form DE/x/y/z, where DE stands for Differential Evolution, x represents the string denoting the solution to be perturbed, y is the number of different solutions to be perturbed and z is the type of crossover used. Two different types of crossover schemas are described; binomial (bin) and exponential (exp) crossover. Binomial crossover stipulates that crossover will occur on each of the *D* values in a solution whenever a randomly generated number between 0 and 1 is within the specified crossover (CR) range. Exponential crossover is performed on the solution until the random value generated between 0 and 1 goes beyond the CR range.

DE Procedure

Differential Evolution uses a 'greedy' selection scheme where the new candidate solutions are generated by combining the selected parent individual and several other randomly selected individuals of the same population. The new candidate thus obtained replaces the parent only if it has better fitness value.

Initial Population

Initial populations of target vectors is generated randomly and the algorithm improves the population iteratively by using mutation, crossover and selection operators.

Mutation

In the mutation operation, a mutant vector is generated using one of the mutation strategies. The strategy DE/rand/1/bin is used in the present work.

Crossover

Crossover controls the amount of diversity of mutant vectors and is used to generate. a trial vector. In this operation, a random number is generated between 0 to 1, and if the random number is less than crossover constant (Cr [0,1]) copy target vector value otherwise mutant vector value.

Selection

The trial vector obtained by the crossover operator is compared with the target vector to determine the member that participates in the next generation. The fittest is passed on to the next generation.

Termination Criteria

The above procedure continues until a stopping criterion is met. This criterion can either have the current best objective function value smaller than a specified value or the number of generations equal to a predetermined maximum value.

Control Parameters

DE performance mainly depends on three parameters: amplification factor of the difference vector – *F*, Crossover control parameter – *Cr* and population size - *NP*. Some guidelines are available to choose the control parameters (). Normally, NP is 5 to 10 times the numbers of parameters in the solution vector. 'F' usually takes a value that ranges from 0.4 to 1.0. F = 0.5 is a good initial choice and it can be increased if the population converges prematurely. On the other hand, a good value for Cr is 0.1 however, to speed up convergence a greater value can be used.

DE Steps

The DE steps are discussed below and the corresponding flowchart is shown in Figure 1.

Step 1: Generate randomly a population of size NP, D- dimensional solution vectors($X_{i.g}$):

$x_{i.g,} = \{ x_{i,1.g}, x_{i,2.g}, \ldots\ldots\ldots\ldots x_{i,D.g} \}$, i =1,2,…*NP*

'g' denotes the generation number.

Step 2: For each population member, *i*, a mutant vector ($U_{i.g+1}$)is formed:

$x_{i,g} \Rightarrow U_{i.g+1} = \{ U_{i,1.g+1}, U_{i,2.g+1}, \ldots\ldots\ldots\ldots U_{i,D.g+1}\}$ i =1,2,…*NP*

Mutant vector is generated using the mutation strategy.

$U_{i.g+1} = x_{r1.g} + F (x_{r2.g} - x_{r3,g})$ (2)

Where $x_{r1.g}$, $x_{r2.g}$ and $x_{r3.g}$ are three distinct random vectors drawn from the population.

Step 3: A crossover is performed for each component of the target/mutant vector to generate

a trail vector($V_{i.g+1}$). Draw a random number called $Rand_j$ between [0,1].

If Randj < Cr then $V_{i,j.g+1}=U_{i,j.g+1}$

else $V_{i,j.g+1}=X_{i,j.g}$ Where i =1,2,…NP and j=1,2,…..D

Step 4: Compare the target vector and trial vector for selection for next generation

Population ($X_{i.g+1}$). If the objective value f($V_{i.g+1}$) is lower than f($X_{i.g}$), then $V_{i.g+1}$ replaces $X_{i.g}$ Otherwise, consider $X_{i.g}$.

Step 5: Iterate the process until the termination criterion is satisfied.

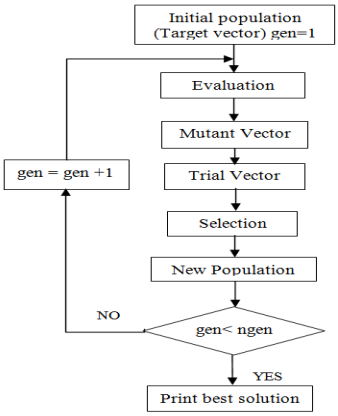

Figure 1: Flow chart of Differential Evolution

Local Search Algorithm

Local search technique is combined with DE for exploration and exploitation among the solutions to obtain a near optimal solution. In this work, a simple swapping mechanism is implemented in which two positions of the vector are selected randomly and swapped. If there is an improvement in the objective function the swap is accepted, otherwise, it is

not considered. Once a swap is accepted, there is a change in the sequence and further improvements in the solutions can be identified by swapping other positions of the vector. The process continues till the search completes for all combinations of swapping. The pseudo-code of local search procedure is given in Figure 2.

Local search (*Current Solution*)

do

{

Current Solution Updated = False

Determine the sequence of *Current Solution*

While

{

New Solution: = swap two random positions in the *Current Solution*

if Makespan (*New Solution*) < Makespan (*Current Solution*) **then**

Current Solution = *New Solution*

Current Solution Updated = **true**

End if

}

Until *Current Solution Updated* = **false**

Return *Current Solution*

Algorithm: Pseudo-Code for Local Search

DE ALGORITHM FOR JOB SHOP SCHEDULING PROBLEM

The natural coding for sequencing and scheduling problem is the permutation vectors (integer vectors). In order to apply DE algorithm, it is crucial to design a suitable encoding scheme that maps the floating point vectors to good permutation vector solutions. In this work, the operation schedule is generated using random keys encoding scheme which deals with floating point vectors (Bean, 1994). In the random keys framework, a permutation of n elements is represented by a vector and this representation transforms a permutation problem to a real-valued optimization problem in the range [0, 1]. Each solution is encoded as a vector of D floating – point members, where D corresponds to the total number of operations in the problem. The components of the vector are sorted in the increasing order and the order in the vector determines the final position. A special method is proposed along with a decoding scheme for finding out the feasible job sequence from the real floating vector. The working of the proposed algorithm is discussed in section 5.1 with a numerical illustration.

Numerical Illustration

To illustrate the proposed algorithm a 3 jobs and 3 machines (3X3) job shop scheduling problem is considered.

The operation sequence and processing times of operations of the jobs on specific machines are given in Table 1. The chromosome representation, random keys encoding mechanism and the decoding procedure are discussed in the following section.

Table 1: Job Set Details

Job. No	M/C	PT	M/C	PT	M/C	PT
1	M2	17	M1	16	M3	18
2	M3	19	M2	14	M1	15
3	M3	14	M1	12	M2	19

Chromosome Representation

Each job contains 3 operations and a total number of operations is 9. All operations of a job are represented by the same number and the interpretation is based on the order of their occurrence in the sequence and is shown in Fig.3. "1" stands for the operations of Job 1 and "2" stands for the operations of job 2 and 3 stands for the operations of Job 3.

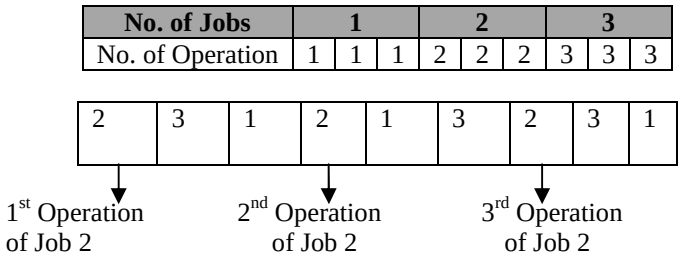

Figure 3: Chromosome Representation

Random Keys Encoding

The total numbers of operations are 9 and hence the floating point vector contains 9 random keys and is shown in Table 2. All random keys in the vector are assigned a rank based on their value. The rank indicates the position of the job in the sequence.

Table 2: Random Keys Encoding

Job	1			2			3		
No. of Operation	1	1	1	2	2	2	**3**	**3**	**3**
Random keys	0.263	0.128	0.625	0.139	0.421	0.212	0.565	0.114	0.232
Assign Rank	6	2	9	3	7	4	**8**	**1**	**5**

Decoding Procedure

In the random keys encoding, if the assigned rank is very low then the corresponding job is given priority and kept first in the sequence. The decoding procedure is shown in Table 3.

Table3: Decoding Procedure

Sort the Ranks in Ascending Order	1	2	3	4	5	6	7	8	9
Operation Sequence (Chromosome)	**3**	1	2	2	**3**	1	2	**3**	1

Initial Population

An initial population (Target vectors) is generated using random keys and is shown in Table 4. The objective value (make span) of each vector is evaluated and the best one is considered as the best solution of the generation.

Table 4: Target Vectors (NP=10)

Vector Number	Random Keys									Make Span
1	0.026316	0.012821	0.125	0.012987	0.04	0.021277	0.055556	0.011111	0.023256	**81**
2	0.014286	0.038462	0.01087	0.0625	0.016129	0.142857	0.01	0.013889	0.011111	**82**
3	0.01087	0.066667	0.010989	0.012821	0.066667	0.03125	0.111111	0.038462	0.076923	**67**
4	0.01	0.052632	0.013158	0.02381	0.020408	0.016393	0.010204	0.047619	0.014706	**81**
5	0.022727	0.022727	0.019608	0.011628	0.066667	0.013333	0.01	0.04	0.012821	**114**
6	0.045455	0.029412	0.011628	0.033333	0.0125	0.027027	0.011111	0.016949	0.019608	**82**
7	0.021277	0.034483	0.011494	0.010526	0.03125	0.012821	0.045455	0.076923	0.01	**81**
8	0.011236	0.015152	0.04	0.013889	0.017544	0.016393	0.055556	0.011494	0.010638	**81**
9	0.030303	0.2	0.071429	0.012346	0.05	0.058824	0.010204	0.013699	0.03125	**106**
10	0.026316	0.015625	0.023256	0.02439	0.018182	0.013699	0.013699	0.015625	0.027027	**93**

Mutant Vector

The target vectors shown in Table 5 are randomly chosen from the initial population to generate corresponding mutant vector using the equation 2. The mutant vectors generated are shown in Table 6.

Table 5: Random Vectors Chosen for Generation of Mutant Vectors

Vector Number	Randomly Chosen Vectors		
	r1	r2	r3
1	1	6	10
2	10	6	9
3	5	6	2
4	6	2	3
5	1	6	4
6	5	10	7
7	6	5	1
8	1	9	6
9	10	6	9
10	6	10	8

Table 6: Mutant Vectors

Vector Number	Random keys								
1	0.03206	0.01696	0.12151	0.01567	0.0383	0.02528	0.05478	0.01151	0.02103
2	0.03086	0.03555	0.00532	0.03069	0.00693	0.00416	0.01397	0.0166	0.02353
3	0.03208	0.02001	0.01984	0.00288	0.06558	0.02142	0.01033	0.04092	0.01537
4	0.04648	0.02095	0.01159	0.04824	0.00266	0.06051	0.01922	0.00958	0.00014
5	0.03695	0.00586	0.12454	0.01584	0.03763	0.02447	0.05583	0.00191	0.02473
6	0.02424	0.01707	0.02314	0.01579	0.06275	0.0136	0.00047	0.02161	0.01793
7	0.04438	0.03238	0.01999	0.03293	0.0205	0.02464	0.00256	0.02562	0.01648
8	0.02177	0.064	0.14294	0.00669	0.05125	0.03082	0.05528	0.01014	0.02675
9	0.03086	0.03555	0.00532	0.03069	0.00693	0.00416	0.01397	0.0166	0.02353
10	0.04998	0.02955	0.00661	0.03648	0.01269	0.02622	0.00145	0.01819	0.02452

Trail Vector

The trail vectors generated using crossover operation are shown in Table 7. The crossover constant Cr=0.35 is used to combine the target and mutant vectors. If the generated random number is below Cr value, then mutant vector gene is copied, otherwise the target vector gene is copied into a trial vector.

Table 7: Trail Vectors

Vector Number	Random keys									Make Span
1	**0.03206**	0.012821	0.125	**0.01567**	**0.0383**	**0.02528**	**0.05478**	0.011111	**0.02103**	81
2	**0.03086**	**0.03555**	**0.00532**	**0.03069**	**0.00693**	0.142857	0.01	0.0166	0.011111	98
3	0.01087	**0.02001**	**0.01984**	0.012821	**0.06558**	**0.02142**	**0.01033**	**0.04092**	**0.01537**	67
4	**0.04648**	**0.02095**	**0.01159**	**0.04824**	0.020408	**0.06051**	**0.01922**	**0.00958**	**0.00014**	79
5	**0.03695**	**0.00586**	**0.12454**	0.011628	**0.03763**	**0.02447**	**0.05583**	0.04	**0.02473**	67
6	**0.02424**	**0.01707**	**0.02314**	0.033333	**0.06275**	**0.0136**	**0.00047**	**0.02161**	**0.01793**	82
7	**0.04438**	**0.03238**	**0.01999**	**0.03293**	**0.0205**	**0.02464**	**0.00256**	**0.02562**	**0.01648**	79
8	**0.02177**	**0.064**	**0.14294**	**0.00669**	0.017544	**0.03082**	**0.05528**	**0.01014**	0.010638	127
9	**0.03086**	**0.03555**	**0.00532**	**0.03069**	**0.00693**	**0.00416**	0.010204	0.013699	**0.02353**	113
10	**0.04998**	**0.02955**	**0.00661**	**0.03648**	0.018182	**0.02622**	**0.00145**	**0.01819**	**0.02452**	82

Best Vectors for Next Generation

The best vectors are generated based on the fitness values of the trail vector and target vector and are shown in Table 8. In comparison, if the trail vector is having better fitness value than that of the target vector, then it is treated as the best vector, otherwise the target vector becomes the best vector and passed to the next generation.

Table 8: Best Vectors Selected for Next Generation Population

Vector Number	Random keys									Make Span
1	0.032057	0.012821	0.125	0.01567	0.038295	0.025275	0.054779	0.011111	0.02103	81
2	0.014286	0.038462	0.01087	0.0625	0.016129	0.142857	0.01	0.013889	0.011111	**82**
3	0.01087	0.020012	0.019835	0.012821	0.065578	0.021416	0.010333	0.040918	0.01537	67
4	0.046479	0.02095	0.011592	0.048237	0.020408	0.060509	0.019222	0.009577	0.000136	79
5	0.036952	0.005855	0.124541	0.011628	0.037628	0.024467	0.055828	0.04	0.024726	67
6	0.024239	0.01707	0.023136	0.033333	0.062746	0.013597	0.000473	0.021611	0.017929	82
7	0.044378	0.032384	0.01999	0.032926	0.0205	0.024644	0.002556	0.025616	0.016477	79
8	0.011236	0.015152	0.04	0.013889	0.017544	0.016393	0.055556	0.011494	0.010638	**81**
9	0.030303	0.2	0.071429	0.012346	0.05	0.058824	0.010204	0.013699	0.03125	**106**
10	0.049978	0.029554	0.006605	0.036484	0.018182	0.026219	0.001446	0.018188	0.024524	82

Results

The DE procedure is iterated for a specified number of generations. Operation schedule and its makespan are presented as the output. The solution vector with decoded operation sequence is shown in Table 9 and the corresponding Gantt chart is shown in figure 4.

Table 9: Operation Sequence

Solution Vector	0.03471	0.20310	0.01109	0.01023	0.02368	0.04985	0.10023	0.03656	0.04590
Operation Sequence	2	1	2	1	3	3	2	3	1

Make Span: 64

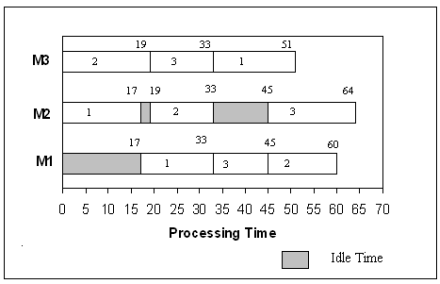

Figure 4: Gantt chart

RESULTS AND DISCUSSIONS

The proposed hybrid DE algorithm is extensively tested on a good number of Job shop scheduling problems (Fisher and Thompson, 1963; Lawrence, 1984; Yamada and Nakano,1992). The performance of DE depends on scaling factor and crossover probability. A scaling factor of 0.65 and a crossover probability of 0.35 is chosen after extensive pilot studies. Different size of job shop scheduling problems with 5, 10 machines and 10, 15, 20 jobs are addressed. The maximum numbers of operations considered in the study are 200. The DE algorithm runs about 20 iterations for each problem to assess the consistency and efficiency of the algorithm. The results of the proposed algorithm are compared with results of Tabu search (Nowicki, 1996), greedy randomized adaptive search procedure (Binato, 2002), and hybrid genetic algorithm (Goncalves,2002), and are shown in Table 10.

Table 10: Comparison of Results

JSSP Instance	Size (n x m)	Best Known Solution	Hybrid Genetic Algorithm	GRASP	Tabu Search	Proposed Algorithm
FT06	6x6	55	55	55	55	**55**
FT10	10x10	930	930	938	930	**930**
FT20	20x5	1165	1165	1169	1165	**1165**
LA01	10x5	666	666	666	666	**666**
LA02	10x5	655	655	655	655	**655**
LA03	10x5	597	597	604	597	**597**
LA04	10x5	590	590	590	590	**590**
LA05	10x5	593	593	593	593	**593**
LA06	15x5	926	926	926	926	**926**
LA07	15x5	890	890	890	890	**890**
LA08	15x5	863	863	863	863	**863**
LA09	15x5	951	951	951	951	**951**
LA10	15x5	958	958	958	958	**958**
LA11	20x5	1222	1222	1222	1222	**1222**
LA12	20x5	1039	1039	1039	1039	**1039**
LA13	20x5	1150	1150	1150	1150	**1150**
LA14	20x5	1292	1292	1292	1292	**1292**
LA15	20x5	1207	1207	1207	1207	**1207**
LA16	10x10	945	945	945	945	**945**
LA17	10x10	784	784	784	784	**784**
LA18	10x10	848	848	848	848	**848**
LA19	10x10	842	842	842	842	**842**
LA20	10x10	902	907	907	902	**902**

CONCLUSIONS

The purpose of this study is an attempt to address the job shop scheduling problem for make span minimization using the differential evolution algorithm. The algorithm is coded in VC++ and is extensively tested on a set of standard job shop scheduling problems. In most of the test cases, it is found that the algorithm was terminated within 200 generations for a scaling factor and crossover probabilities of 0.65 and 0.35 respectively. The algorithm is also tested with hybridization using neighborhood, local search and is observed that the results are converged at a faster rate than the pure DE algorithm. The results are compared with the best-known solutions reported in the literature and is observed that the proposed approach is performing well in all the benchmark problems tested.

REFERENCES

1. *Andreas C. Nearchou and Sotiris L. Omirou. (2006). Differential Evolution for sequencing and scheduling optimization. J Heuristics, 12: 395 – 411.*
2. *Applegate. D and W.Cook. (1991). A Computational study of Job shop scheduling instance. ORSA Journal on Computing,3 : 149-156*
3. *Bagchi, T.P, (1999). Multi objective scheduling by Genetic algorithm. Kluwer Academic Publishers.*
4. *Baker, K. R, (1974). Introduction to Sequencing and Scheduling. John Wiley, New York.*
5. *Bean,J.C.(1994). Genetic algorithms and random keys for sequencing and optimization. ORSA journal on computing, 6:154-159.*
6. *Biegel, J. E., and J.J Davern. (1990). Genetic algorithms and job shop scheduling. Computers and Industrial Engineering 19: 81-91.*
7. *Binato, S., Hery, W.J., Loewenstern, D.M and M.G.C Resende. (2002). A GRASP for job shop scheduling. In: Essays and Survey in Meta heuristics, Riberio, Celso C., Hansen, Pierre (Eds), Kluwer Academic Publishers.*
8. *Carlier, J. and E. Pinson. (1989). An algorithm for solving the job-shop problem. Management Science 35:164-176.*
9. *Cheng, R., Gen, M., Y. Tsujimura. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms-I representation. Computers and Industrial Engineering 30(4): 983-997.*
10. *Falkenauer E. and S. Bouffouix. (1991).A Genetic algorithm for job shop. In: The Proceedings of the IEEE International Conference on Robotics and Automation, Sacremento, California, USA.*
11. *Fisher. H, and G.L Thompson. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In: Industrial Scheduling, J.F Muth and G.L Thompson (pp.225-251), Prentice Hall, Englewood Cliffs, New Jersey.*
12. *Goldberg, D.E, (1981). Genetic Algorithms in search, optimization & machine learning. Addison- Wesley.*
13. *Goncalves, J. F, Mendes, J.M and G.C Resende. (2002).A Hybrid genetic Algorithm for Job-shop scheduling problem. Technical report TD-5EAL6J. AT & T Labs.*
14. *Hart W.E, Krasonogor N, J.E Smith. (2004). Recent advances in memetic algorithms. Springer, Heidelberg, New York.*
15. *Yadav, Rahul. "Message Security using Cryptography and Lsb Algorithm of Steganography."*
16. *Jain, A.S and Meeran, S., (1999), Deterministic job-shop scheduling: Past, Present and future. European Journal of Operations Research, 113: 390-434.*
17. *Kaelo, P and M.M. Ali. (2006). A numerical study of some modified differential evolution algorithms. European Journal of Operational Research, 171:674 – 692.*
18. *Lawrence, S. (1984). Resource constrained project scheduling: An Experimental Investigation of Heuristic Scheduling techniques. GSIA, Carnegie Mellon University, Pittsburg, PA.*
19. *Nowicki, E. and C.Smutnicki. (1996). A fast Tabu search algorithm for job shop problem. Management Science,42(6): 797-813.*
20. *Storn R, and K.Price. (1997).Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 2:341-359.*

21. *Yamada, T and R.Nakano. (1992). A genetic algorithm applicable to large scale job shop problems. In:Manner, R.,B.Manderick.(Eds.), Proceedings of the Second International workshop on Parallel problem solving from Nature (pp. 281-290). Brussels, Belgium.*